

Logic Programming Engineering – Assignment 9

Dr.-Ing. Sascha Klüppelholz

Exercise 9.1 [Word guessing game]

The goal of this exercise is to implement a word guessing game, based on a randomly chosen word from a knowledge base. In Prolog the knowledge base should be implemented as a dynamic predicate.

Besides implementing the game itself, your Prolog program should allow for maintaining the knowledge base as well as providing file management functionalities. In particular the following features should be accessible via a menu: adding/deleting a word to/from the knowledge base, list all words in the knowledge base, reading/writing a knowledge base from/to a database file (replacing the old), start playing the game, and ending the program.

The game: The aim of the game is for a player to guess an unknown word with a minimal number of guesses. For this, a random word from the knowledge base is chosen. At the beginning of the game each letter of the selected word is replaced by a *-character and shown to the player. In each round of the game the player guesses one letter. If this letter is contained in the word, the *-character at the corresponding positions are replaced with the character. The game continues until the full word is visible.

Example:

```
G u e s s   a   W o r d
-----
r - read database file
l - list knowledge base
a - add a new word
d - delete a word
w - write database file
g - guess a word
e - end the game
Command: g

Please guess the word: ***
Please guess a letter: a
Your solution:         *a*
Please guess a letter: d
Your solution:         *a*
Please guess a letter: t
Your solution:         *at
Please guess a letter: c
Your solution:         cat

Congratulations! It took you only 4 guesses.
```

For this exercise, work in groups and split the project in subtasks, e.g.,

menu structure and processing user commands: the menu should appear automatically when the program is consulted. A single character should be taken as an input (without pressing enter). The corresponding command should then be executed and after processing the given command (other than end-of-game), the menu should again be displayed.

Useful predicates for this subtask are, e.g., the control predicate `repeat/0`, primitive character I/O-predicates such as `get_single_char/1` and `put/1`, predicates for term reading and writing such as `print/1` and `read/1`, and predicates for analyzing and constructing atoms, e.g., `atom_codes/2` (or `name/2`).

maintaining the database: the knowledge base should be represented as facts, using a dynamic predicate `w/1`, where the fact `w(word)` means that the word is contained in the knowledge base. This subtask covers adding to and deleting words from the knowledge base as well as the reading and writing (replacing) of database files. Please make sure that you carry out all necessary checks, e.g., avoid having duplicates in the knowledge base, warnings at unsaved changes in the knowledge base, file-not-found errors on reading, warning when overwriting existing files. Initially the database should be empty. Whenever a new database is loaded from a file (for which the user should provide a filename) the old words are to be erased beforehand. The database file itself consists of the words in the database, each word in a separate line in the input file. For maintaining the database, the following predicates might be helpful: `dynamic/1`, `assert/1`, `retractall/1`. For reading and writing streams you may consider using, e.g., the predicates `open/3`, `read/2`, `close/1`.

game playing: this subtask covers the game playing. A random word should be selected from the knowledge base, by collecting all words from the knowledge base in one set and picking one randomly. After that, a *-character representation should be created and the guessing is started. For the game playing procedure it might be helpful converting the given word into a list of characters. The players guesses should be recorded and processed until the full word is guessed. The number of guesses are to be recorded and printed at the successful end of the guessing game.

Useful predicates for this subtask are, e.g., primitive character I/O-predicates such as `get_single_char/1`, predicates for term reading and writing such as `print/1`, and predicates for analyzing and constructing atoms, e.g., `atom_chars/2`.

At the very end, compose all predicates into a single program providing the requested functionality.